xAltarica

Christophe de Charentenay, ENSAE, Toulouse, octobre 2003

Le monde XML

xAltarica consiste en une spécification du langage altarica en langage XML. Altarica rejoint alors le monde des langages XML, comme MathML, HTML, ou xADL, etc…. Les modèles altarica seront formatés selon la norme XML et seront conformes aux spécifications xAltarica.

Cette démarche qui ne modifie pas la sémantique d'altarica, lui apporte un certain nombre de moyens : spécification du langage, accès aux données, manipulation des données, intégration du langage dans les applications, et qui, somme toute, devrait fortement améliorer deux propriétés non fonctionnelles du langage : son extensibilité et son interopérabilité avec d'autres langages.

Ces moyens sont à la fois, des process, des techniques normalisées et des implémentations, comme l'indique le tableau (Cette liste est retreinte)

Process	*Technique*	*API/Outils supports*
Spécification du langage	XML schema	XML spy
Parsing et validation syntaxique des documents	Parseurs XML : SAX, DOM	Xerces, JDOM, JAXP
Transformation des documents	XSL-XSLT	Xalan, Saxon, JAXP
Accès aux données du documents	XPath, XQuery	Qizx, Saxon
Intégration du langage	XML Java Data Bnding	JAXB, XML spy

Les différents process et techniques utilisés méritent quelques explications :

Spécification XML du langage

XML offre deux méta langages de spécification de structure de document (dans le cas présent la structure de document équivaut au langage) , la DTD et XML Schema. Sans entrer dans les détails, signalons simplement qu'XML Schéma est apparu après les DTD, et qu'il apporte des mécanismes de typage, ce qui lui confère quelques avantages quant à l'extensibilité du langage et la puissance de validation des documents. De plus XML Schema est lui même un langage XML

XML schema permet de hiérarchiser le langage afin de distinguer la part générique des extensions spécifiques du langage. Enfin les extensions postérieures ne modifient pas les spécifications antérieurs, permettant ainsi d'éviter les phénomènes de régression.

xAltarica se compose actuellement de trois schémas définissant trois namespaces :

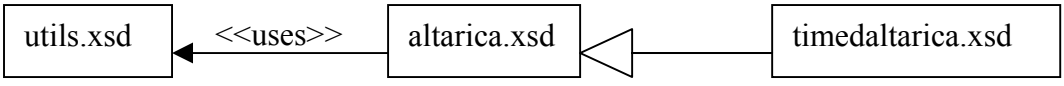

Figure 1 : Les schémas d'xAltarica

- **altarica.xsd** : Architecture, ComplexComponent, Sub, Sync, ConfigurationAssert, Component, Flow, Event, State, Transition, ComputationAssert, Guard …
- **utils.xsd** : ArithmeticExpression, ArithmeticValue, ComparisonExpression, BooleanExpression, BooleanValue, Constant, Variable, Value, Pointer…
- **timedAltarica** : (en cours d'étude) TimedComponent, …

utils.xsd rassemble les outils mathématiques comme les expressions
altarica.xsd qui utilise les termes d'utils xsd sert à représenter les modèles altarica
timedAltarica.xsd, à l'état de développement potentiel, étend les termes d'altarica.xsd pour intégrer les attributs temporels.

La spécification proposée ne modifie en rien la sémantique d'altarica, quelques modifications de lexique et de syntaxe y ont été introduites pour des raison de meilleure compatibilité avec les langages d'architecture, en particulier xADL. Ainsi, on observe que Node est décomposé en Architecture, ComplexComponent et Component et Assert en Configuration Assert et ComputationAssert. Ceci permet de distinguer d'une part les composants atomiques des composants composites, d'autre part le comportement dynamique de la configuration.

Parsing et Validation des documents XML

C'est l'un des avantages majeurs, que permet de formatage des documents en XML : bénéficier des parseurs génériques XML, ceci, quelque soit les spécifications de structure. Ces parseurs s'intègrent facilement dans les environnements support et réduisent d'autant l'effort de développement des applications. Ils sont génériques et acceptent sans modification toute évolution du langage. Les parseurs XML génériques permettent en outre la vérification et la validation syntaxique pour autant qu'il soit indiqué dans l'entête du document à quels schémas ce dernier est supposé conforme.

On distingue deux types de parseurs XML :
Les parseurs SAX qui fonctionnent en read only et parcourent le document en séquence. Ils sont très adaptés aux gros documents (ce qui ne devrait pas être notre cas)
Les parseurs DOM qui construisent une représentation interne du document sous forme d'arbre syntaxique. Ce type de parseur sert aussi en écriture pour la sérialisation des données en document xml. Le style DOM est bien adapté aux traitements et à la manipulation des documents.

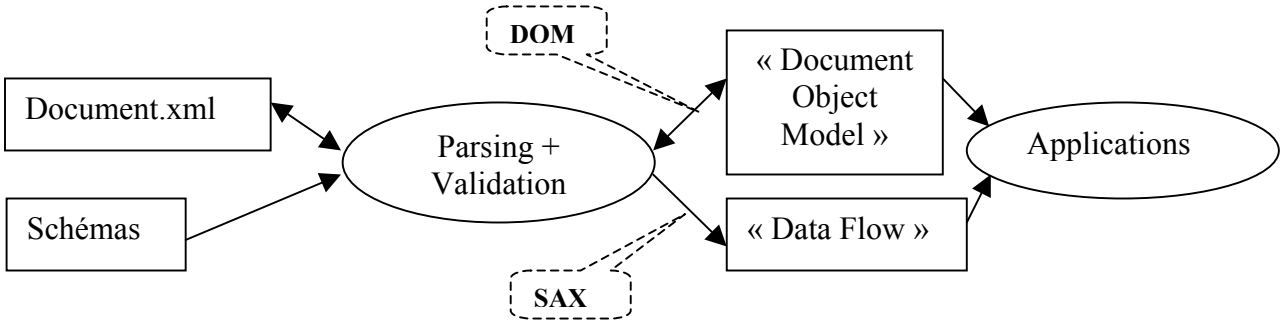

Figure 2 : Parsing et Validation

Transformation d'un format de donnée à un autre

La transformation d'un document XML permet de reformater le document pour des besoins de traduction ou de présentation. Le principe consiste à transformer un arbre source représentant le document sous sa forme de départ en arbre résultat conforme au format de destination désiré. La spécification de la transformation se trouve dans un document appelé feuille de style suivant un langage appelé XSL. Cette feuille de style permettra au processeur de transformation générique (XSLT) d'opérer la transformation sur le document.

L'usage d'XSL est multiple, il s'utilise beaucoup pour la mise en page de document, mais il est aussi utilisé pour permettre l'interopérabilité entre deux langages XML, pour peu qu'ils soient compatibles.

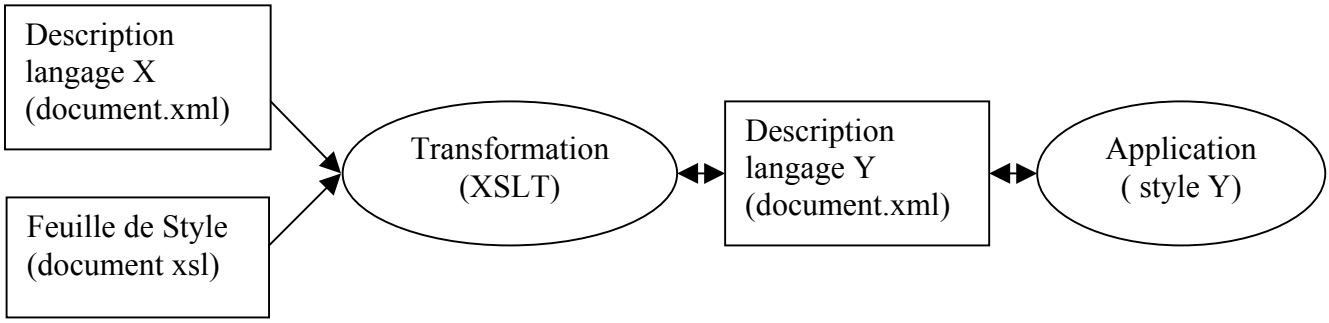

Figure 3 : Transformation XSLT

XML Java Data Binding : de la spécification XML à l'implémentation objet

Le Java Data Binding entre XML et Java consiste à dériver depuis les schémas des classes et des interfaces équivalentes implémentées en java. Celles-ci vont être ensuite utilisées pour construire une représentation interne du document xml avec des objets représentatifs du lexique plutôt qu'avec les objets du DOM (element, attribut, text, …).

Le databinding s'effectue en deux temps :

- La compilation des classes : cette compilation est automatisée et permet de générer une librairie de classes et d'interfaces directement dérivées des schémas.
- La génération et la manipulation d'un modèle composé d'objets instances de ces classes, généré à partir du document xml, et inversement.

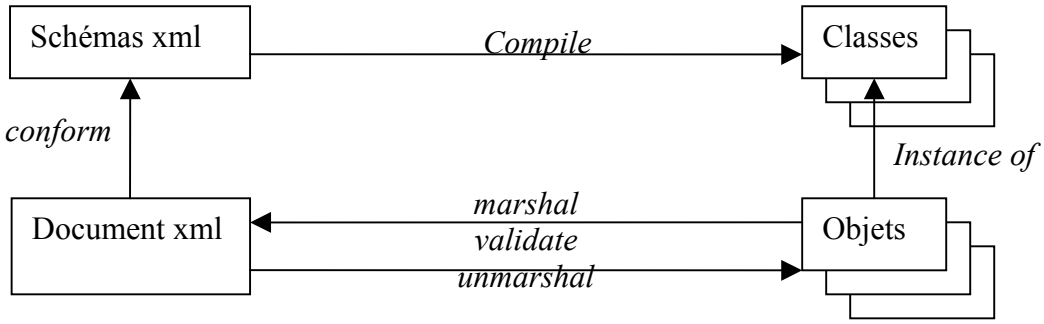

Figure 4 : mapping XML - java (source JAXB specification)

L' architecture JAXB

A - Génération et compilation des Classes et des Interfaces

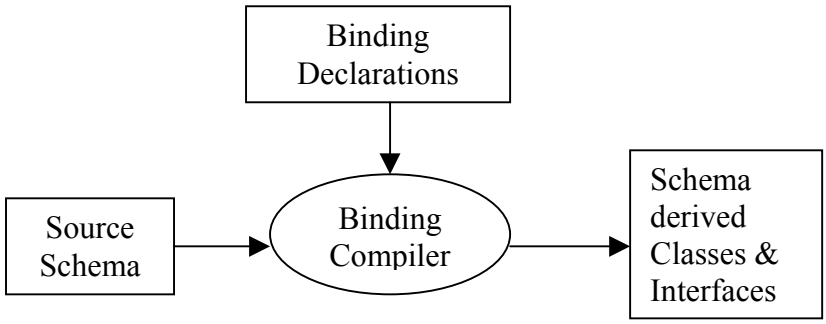

Figure 5 : Binng Compilation

Les « binding declarations » servent à « customiser » le mapping entre xml schema et java, ces déclarations sont optionnelles (il existe un mapping par défaut). Elles peuvent être contenues dans un fichier externe ou à l'intérieur même des schémas.

Mapping XML to Java (extrait) :

XML Schema	Java représentation
Schema	Package
ComplexType	Content Interface
Derived by Extention	Extends
Attribute	Property (Accesseurs, modifieurs)
Element	Property (Accesseurs, modifieurs)

Figure 6 : mapping XML Schema - Java representation

B – Binding document/modèle objet, manipulation du modèle

Les interfaces et leur implémentations crées permettent alors de construire puis de manipuler une représentation interne du document en regard du langage.

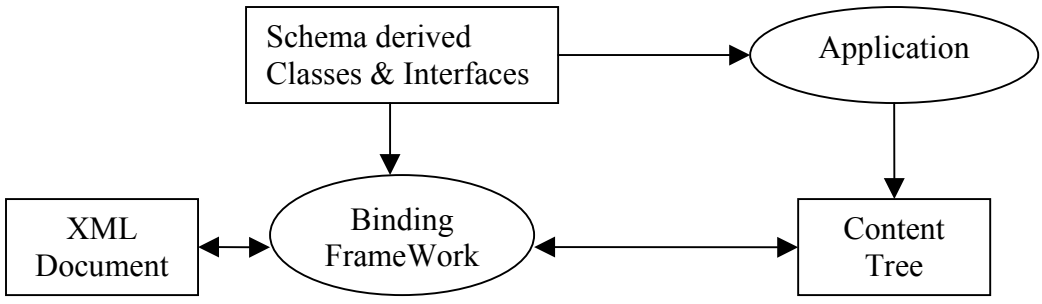

Figure 7 : Binding Framework

Impact sur l'évolution du langage

L'utilisation d'xml, d'xml schema, d'xsl et éventuellement du data binding facilite l'évolution future d'altarica :

- XML schema permet d'étendre le langage facilement, sans régressivité
- Les nouvelles extensions sont sans effet sur les outils génériques tels les parseurs SAX, DOM, JDOM, les processeurs XSLT ou encore les compilateurs de Data Binding.
- Les API's « syntax directed » s'adaptent de facto aux évolutions.
- Les librairies dérivées des schémas sont mises à jour par simple recompilation.

Ainsi, le travail de développement et d'adaptation des outils peut se concentrer sur l'essentiel des évolutions linguistiques, leur sémantique, et leur traitement sémantique.

Impact sur l'interopérabilité

L'interopérabilité bénéficie aussi de ces techniques pour la mise en œuvre :

L'interopérabilité peut être interne entre plusieurs vues contenues dans la sémantique générale du langage :

- par une utilisation modulaire du langage.

Elle peut être externe, avec d'autre langage (ou plus généralement d'autres formats), dans ce cas elle s'opère

- au moyen d' XSL et XSLT
- ou par intégration du Data Binding sur plusieurs environnements, et/ou transformation par des API spécifiques

Exmple d'interopérabilité :

Modèle xAltarica, modèle xADL

Les deux langages n'ont pas le même point de vue, ni la même finalité. xAltarica va permettre de décrire les composants par leurs états de fonctionnement, et la dynamique de ces états. La configuration architecturale est implicite. Dans un autre registre xADL permet de spécifier les architectures sous l'angle de la configuration et de l'implémentation et ne s'occupe pas d'aspects comportementaux analysables.

Toutefois xADL s'est construit depuis quelque années avec un souci de généricité et d'ouverture, en conservant la possibilité d'être étendu facilement à d'autres fins comme l'analyse de propriétés. Ces extensions restent à réaliser.

Altarica et xADL possède en commun la configuration architecturale (implicite chez l'un, explicite et de premier ordre chez l'autre)

Altarica possède en propre la capacité à décrire l'état de fonctionnement des composants et permet ensuite les études de SdF

xADL insiste sur le design architectural en offrant le typage explicite des composants. Il permet la description de familles d'architectures, avec options et variantes (*PLA, Product Line Architecture*). Il est bien adapté à la génération de code applicatif par assemblage de composants (implémentés)

C'est en effectuant le mapping détaillé entre les deux langages que l'on peut évaluer la possibilité et la portée de l'interopérabilité.

xAltarica	xArch
	Description
Pointer	XMLLink
(attr) xlink :type	(attr) xlink :type
(attr) xlink :href	(attr) xlink :href
Architecture/ComplexComponent	ArchInstance
(attr) Name	(attr) Identifier
	Description
Component / ComplexComponent	ComponentInstance
derived from ConfigurationAssert, Sync	ConnectorInstance
derived from ConfigurationAssert, Sync	LinkInstance
	Group
Component | ComplexComponent | *Sync*	ComponentInstance
(attr) Name	(attr) Identifier
	Description
Event/Flow	InterfaceInstance
from Sub, ConfigurationAssert	SubArchitecture
from ConfigurationAssert, Sync	ConnectorInstance
	(attr) Identifier
	Description
Event/Flow	InterfaceInstance
	SubArchitecture
Event/Flow	InterfaceInstance
(Attr) Name	(attr) Identifier
	Description
Way	Direction
	SubArchitecture
derived from Sub, SubLink	ArchInstance
HierarchicLink	InterfaceInstanceMapping
HierarchicLink	InterfaceInstanceMapping
flowParent : FlowPointer	outerInterfaceInstance : XMLLink
flowInstanceChild : FlowInstance	innerInterfaceInstance : XMLLink
ConfigurationAssert	LinkInstance
	Description
FlowInstance	Point
FlowInstance	Point
SubComponantPointer & FlowPointer	anchorOnInterface : XMLLink

Figure 8 : Mapping xAltarica/xADL2.0

Le mapping ci dessus est dans le sens xAltarica vers xArch . xArch est en fait le schéma de base d'xADL2.0, le mapping complet devrait inclure les autres schémas, notamment « type.xsd » qui permet de typer les composants et de séparer leur design de leurs instanciations dans des stuctures ou des sousstructures.

Conclusion

xAltarica est avant tout un outils pour les outils support d'altarica. En permettant l 'accès aux technologies XML, les outils deviennent plus adaptables et le langage y gagne en capacité à évoluer et en interopérabilité.

Perspectives

Toutefois, à l'heure qu'il est, cette proposition en est encore au stade initial, il s'agit pour la pousuivre :

- D'en évaluer l'intérêt
- De finaliser une première version d'xAlatarica
- De réaliser les extensions comme timedAltarica
- D'adapter les outils à XML, en envisageant la possibilité d'utiliser le Data Binding
- D'étudier le mapping avec d'autres langages